TÉCNICO LISBOA

Deep Learning

MEEC, MEIC, MECD

Final Exam

January 22, 2025

Version A

Instructions

- You have 120 minutes to complete the exam.
- Make sure that your test has a total of 13 pages, then write your full name and student number on this page (and your number in all the other pages).
- The test has a total of 20 questions, with a maximum score of 100 points. The questions have different levels of difficulty. The point value of each question is provided next to the question number.
- Please provide your answer in the space below each question. If you make a mess, clearly indicate your answer.
- The exam is open book and open notes. You may use a calculator, but any other type of electronic or communication equipment is not allowed.
- Good luck.

Part 1	**Part 2**	**Part 3, Pr. 1**	**Part 3, Pr. 2**	**Total**
32 points	18 points	25 points	25 points	100 points

Part 1: Multiple Choice Questions (32 points)

In each of the following questions, indicate your answer by choosing **one single** option. Read all the options carefully before choosing.

1. (4 points) The softmax transformation may be modified to include a temperature-like parameter: temp-softmax$_T(\boldsymbol{z})$ = softmax$(\boldsymbol{z}/T)$, where softmax denotes the standard softmax and $T > 0$ is the so-called temperature. Which of the following statements is **false**?

   - ■ **For all $\boldsymbol{z}$, there is no finite value of T for which temp-softmax$_T(\boldsymbol{z})$ is a uniform distribution.**
   - □ For any $\boldsymbol{z}$, as $T \to \infty$, temp-softmax$_T(\boldsymbol{z})$ approaches a uniform distribution.
   - □ For $\boldsymbol{z} = (4, 3, 2, 1)$, as $T \to 0$, temp-softmax$_T(\boldsymbol{z})$ approaches $(1, 0, 0, 0)$.
   - □ For any $\boldsymbol{z}$ and any $T > 0$, temp-softmax$_T(\boldsymbol{z})$ has no zero components.

   Solution: For $\boldsymbol{z} = (a, a, a, a)$, for any real value a, temp-softmax$_T(\boldsymbol{z})$ = (1/4, 1/4, 1/4, 1/4), for any real value T.

2. (4 points) Consider a linearly separable dataset $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$ with features in $\mathbb{R}^D$ and binary labels in $\{-1, +1\}$. Linear separability implies that there is some $\boldsymbol{w}_* \in \mathbb{R}^D$ such that

   $$\forall (\boldsymbol{x}, y) \in \mathcal{D}, ((y = 1) \Rightarrow (-\boldsymbol{w}_*^\top \boldsymbol{x} < 0)) \ \wedge \ ((y = -1) \Rightarrow (\boldsymbol{w}_*^\top \boldsymbol{x} < 0)).$$

   Consider a logistic regression model $\mathbb{P}(y = +1 \,|\, \boldsymbol{x}) = \frac{1}{1+\exp\{-\boldsymbol{w}^\top \boldsymbol{x}\}}$, assuming that the first component of every feature vector $\boldsymbol{x}$ is 1 to account for the bias. Then, the cross-entropy loss

   $$\mathcal{L}(\boldsymbol{w}) = \sum_{y_i=1} \log(1 + \exp\{-\boldsymbol{w}^\top \boldsymbol{x}_i\}) + \sum_{y_i=-1} \log(1 + \exp\{\boldsymbol{w}^\top \boldsymbol{x}_i\}),$$

   where $\sum_{y_i=1}$ and $\sum_{y_i=-1}$ denote the sum through all positive and negative examples, respectively, has

   - □ one global minimum.
   - ■ **no global minima.**
   - □ several global minima.
   - □ several local minima.

   Solution: The loss $\mathcal{L}(\boldsymbol{w})$ is bounded below by 0 because $\log(1 + \exp(u)) > 0$ for any $u \in \mathbb{R}$. It is clear that $\mathcal{L}(\alpha \boldsymbol{w}_*)$ is a strictly decreasing function of α that satifies $\lim_{\alpha \to \infty} \mathcal{L}(\alpha \boldsymbol{w}_*) = 0$, thus $\mathcal{L}$ has no global minima.

3. (4 points) Consider a logistic regression model $\mathbb{P}(y = +1 \,|\, \boldsymbol{x}) = \frac{1}{1+\exp\{-\boldsymbol{w}^\top \boldsymbol{x}\}}$, where $\boldsymbol{x} \in \mathbb{R}^D$ and $y \in \{0, 1\}$, where it is assumed that the first component of every feature vector $\boldsymbol{x}$ is 1 to account for the bias. Consider the following loss:

   $$\mathcal{L}(\boldsymbol{w}) = -y_i \log(\mathbb{P}(y_i = +1 \,|\, \boldsymbol{x}_i)) - (1 - y_i) \log(1 - \mathbb{P}(y_i = +1 \,|\, \boldsymbol{x}_i)) + \frac{\lambda}{2} \|\boldsymbol{w}\|_\infty^2$$

   This is a variant of the cross-entropy loss that penalizes the squared ∞-norm, which is defined as $\|s\|_\infty = \max\{|s_1|, \ldots, |s_D|\}$, for $s \in \mathbb{R}^D$. Let argmax$((s_1, \ldots, s_D))$ denote the index with the highest entry of $(s_1, \ldots, s_D)$ (i.e., argmax$((1, 5, 3)) = 2$) and let $\boldsymbol{e}_i$ denote the i-th canonical basis vector (i.e., $\boldsymbol{e_2}$ in $\mathbb{R}^4$ is $(0, 1, 0, 0)$). The stochastic gradient descent rule for a learning rate of η is

- ■ $w \leftarrow w - \eta\left((\mathbb{P}(y=+1\,|\,\boldsymbol{x}) - y)\,x + \lambda \boldsymbol{w} \odot \boldsymbol{e}_{\text{argmax}((|w_1|,\ldots,|w_D|))}\right)$
- □ $w \leftarrow w - \eta\left((\mathbb{P}(y=+1\,|\,\boldsymbol{x}) - y)\,x + \lambda \boldsymbol{w} \odot \boldsymbol{e}_{\text{argmax}((|x_1|,\ldots,|x_D|))}\right)$
- □ $w \leftarrow w - \eta\left((\mathbb{P}(y=+1\,|\,\boldsymbol{x}) - y)\,x + \lambda \boldsymbol{e}_{\text{argmax}((|w_1|,\ldots,|w_D|))}\right)$
- □ $w \leftarrow w - \eta\left((\mathbb{P}(y=+1\,|\,\boldsymbol{x}) - y)\,x + \lambda \boldsymbol{e}_{\text{argmax}((|x_1|,\ldots,|x_D|))}\right)$

Solution: We have that

$$\frac{\partial}{\partial \boldsymbol{w}}\left(-y_i \log(\mathbb{P}(y=+1\,|\,\boldsymbol{x})) - (1-y)\log(1-\mathbb{P}(y=+1\,|\,\boldsymbol{x}))\right) = (\mathbb{P}(y=+1\,|\,\boldsymbol{x}) - y_i)\,x.$$

It remains to specify $\frac{\partial}{\partial \boldsymbol{w}}\|\boldsymbol{w}\|_\infty^2$. Now,

$$\begin{aligned}
\|w\|_\infty^2 &= (\max(|w_1|,\ldots,|w_D|))^2 \\
&= \max(|w_1|^2,\ldots,|w_D|^2) \\
&= \begin{cases} w_1^2, & w_1 \geq w_j,\ \forall j \neq 1 \\ \ldots & \\ w_D^2, & w_D \geq w_j,\ \forall j \neq D \end{cases},
\end{aligned}$$

which in turn implies that:

$$\begin{aligned}
\frac{\partial}{\partial w_i}\|\boldsymbol{w}\|_\infty^2 &= \begin{cases} 2w_i, & w_i \geq w_j,\ \forall j \neq i \\ 0, & \text{otherwise} \end{cases} \\
&= \begin{cases} 2w_i, & i = \text{argmax}((|w_1|,\ldots,|w_D|)) \\ 0, & \text{otherwise} \end{cases} \\
\Rightarrow \frac{\partial}{\partial \boldsymbol{w}}\|\boldsymbol{w}\|_\infty^2 &= 2\boldsymbol{w} \odot \boldsymbol{e}_{\text{argmax}((|w_1|,\ldots,|w_D|))}.
\end{aligned}$$

Finally, we get:

$$w \leftarrow w - \eta\left((\mathbb{P}(y=+1\,|\,\boldsymbol{x}) - y)\,x + \lambda \boldsymbol{w} \odot \boldsymbol{e}_{\text{argmax}((|w_1|,\ldots,|w_D|))}\right)$$

4. (4 points) Consider a convolutional layer with a 3×5 filter with a stride of 1 in the first dimension, a stride of 3 in the second dimension, and no padding. If the input feature map has size 128×128, what will be the dimensions of the output feature map?
   - □ 126×124
   - ■ 126×42
   - □ 42×126
   - □ 42×42

   Solution: Simply apply the expression $M = (N - F)/S + 1$ to rows and columns.
   Rows: $(128 - 3) + 1 = 126$. Columns: $(128 - 5)/3 + 1 = 42$.

5. (4 points) Which of the following statements is most accurate regarding the latent space of a dense auto-encoder?
   - □ It is a high-dimensional space where the input data is explicitly mapped, usually with more dimensions than the input.

- ☐ It is a space where the encoder attempts to transform the input to be linearly separable for classification purposes.
- ■ **It is a lower-dimensional space that aims to capture the most salient features of the input, enabling a compressed representation.**
- ☐ It is a space where data points are perturbed with noise during training to improve the robustness of the model.

6. (4 points) Consider a recurrent neural network (RNN) that takes as input any sequence of real-valued scalars $\boldsymbol{x} = (x_1, \ldots, x_T)$ and predicts a single output $y \in \mathbb{R}$. The RNN updates its hidden states using the following equations:

$$\begin{aligned} h_0 &= 0, \\ z_t &= w h_{t-1} + a x_t, \\ h_t &= g(z_t), \end{aligned}$$

where $a, w \in \mathbb{R}$ are learnable parameters and g is a predefined activation function. The predicted output of the RNN corresponds to the last hidden state, that is, $\hat{y} = h_T$.

Consider a sequence with two elements $\boldsymbol{x} = (x_1, x_2)$, the corresponding output y, and the squared loss $\mathcal{L}(y, \hat{y}) = \frac{1}{2}(\hat{y} - y)^2$. For a learning rate η, the stochastic gradient descent (SGD) update rule for w is

- ☐ $w \leftarrow w - \eta\,(h_2 - y)\,g'(z_2)\,(x_2 + w\,g'(z_1)\,x_1)$
- ■ $w \leftarrow w - \eta\,(h_2 - y)\,g'(z_2)\,h_1$
- ☐ $w \leftarrow w - \eta\,(h_2 - y)\,g'(z_2)\,g'(z_1)$
- ☐ $w \leftarrow w - \eta\,(h_2 - y)\,g'(z_2)\,w$

Solution: We have

$$\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(g(w\,g(w\,h_0 + a\,x_1) + a\,x_2) - y)^2 = \frac{1}{2}(g(w\,g(a\,x_1) + a\,x_2) - y)^2.$$

Computing the derivative with respect to w, we get

$$\begin{aligned} \frac{d\mathcal{L}(\hat{y}, y)}{dw} &= (\underbrace{g(w\,g(a\,x_1) + a\,x_2)}_{h_2} - y)\frac{d}{dw}(g(w\,g(a\,x_1) + a\,x_2)) \\ &= (h_2 - y)\,g'(\underbrace{w\,g(a\,x_1) + a\,x_2}_{z_2})\,\frac{d}{dw}(w\,g(a\,x_1) + a\,x_2) \\ &= (h_2 - y)\,g'(z_2)\,\underbrace{g(a\,x_1)}_{h_1} \\ &= (h_2 - y)\,g'(z_2)\,h_1, \end{aligned}$$

and so the SGD update is $w \leftarrow w - \eta\,(h_2 - y)\,g'(z_2)\,h_1$.

7. (4 points) Which of the following statements is **true**?

- ☐ The training forward pass of a sequence can be parallelized in an encoder-decoder RNN.
- ☐ The test forward pass of a sequence can be parallelized in an encoder-decoder RNN.

- ■ **The training forward pass of a sequence can be parallelized in a decoder transformer.**
- □ The test forward pass of a sequence can be parallelized in a decoder transformer.

8. (4 points) In a transformer architecture, which of the following is **not** a typical characteristic of the multi-head attention mechanism?
   - □ It involves multiple sets of query, key, and value projections to allow the model to attend to different aspects of the input.
   - □ It scales the dot product attention scores by the square root of the dimension of the keys to stabilize training and prevent vanishing gradients.
   - ■ **It computes a single weighted average of the value vectors, using attention weights calculated from the dot product of queries and keys.**
   - □ It concatenates the results of individual attention heads, followed by a linear transformation to produce a final output for the layer.

Part 2: Short Answer Questions (18 points)

Please provide **brief** answers (1-2 sentences) to the following questions.

1. (6 points) In optimization, when using stochastic gradient descent, what tradeoffs should be taken into account when choosing the minibatch size?

   Solution: Small minibatches lead to faster updates, more exploration of the loss landscape (can help escape local minima) but yield noisier gradient estimates (less accurate estimate of the global gradient), and may require smaller learning rate. Large minibatches provide more accurate gradient estimates, potentially faster convergence in terms of iterations, but are slower to compute with higher memory usage, and higher risk of getting stuck in local minima. Often, in practice, the batch size is chosen to exhaust the memory of the GPU.

2. (6 points) What is a major drawback of both the logistic sigmoid and tanh activation functions, and what activation function is a good workaround?

   Solution: Sigmoid and tanh activation functions cause vanishing gradients, as gradients go to zero exponentially for large positive or negative inputs. The ReLU activation function mitigates this issue by allowing gradients to propagate for positive inputs.

3. (6 points) Explain briefly what skip connections (also known as residual connections) are and why they are used.

   Solution: Skip connections are shortcuts in neural networks that allow data to flow past some layers. This helps gradients flow better in backpropagation and allows the network to reuse features learned in earlier layers. They also act as a safeguard against overfitting since they can be seen as a form of regularization that encourages the network to learn simpler features.

Part 3: Problems (50 points)

Problem 1: Convolutional Neural Networks (25 points)

Problem 1: Convolutional neural networks (5 points) Street-view house numbers (SVHN) is a real-world image dataset for developing machine learning and object recognition algorithms. It can be seen as similar in flavor to MNIST (e.g., the images are of small cropped digits), but incorporates an order of magnitude more labeled data (over 600,000 digit images) and comes from a significantly harder real world problem (recognizing digits and numbers in natural scene images). SVHN is obtained from house numbers in Google Street View images.

Figure 1: Sample images from the SVHN dataset.

In this exercise, assume that the SVHN dataset has been preprocessed by converting the images to grayscale. Thus, each image $\boldsymbol{x}$ in the dataset is a 32×32 grayscale image. Each image

is associated with a given class label $y \in \{0, 1, \ldots, 9\}$, corresponding to the digit that is in the center of the image.

Consider the convolutional neural network depicted in Fig. 2, which is used to predict the class $\hat{y}$ associated with a given input image $\boldsymbol{x}$. The convolutional neural network interleaves convolutional layers with average pooling layers, followed by a linear mapping to produce the logits associated with each class. An average pooling layer calculates the average value for different patches of a feature map and uses it to create a downsampled (pooled) feature map. Also, the average pooling layer operates over each channel independently. The neural network comprises the following layers:

- *Conv-1*: convolutional layer with 8 filters of kernel size 5×5, stride 1 and padding of 0.
- *AvgPool-1*: average pooling layer with a kernel size of 4×4, stride 4 and padding of 0.
- *Conv-2*: convolutional layer with 16 filters of kernel size 4×4, stride 1 and padding of 1.
- *AvgPool-2*: average pooling layer with a kernel size of 2×2, stride 2 and padding of 0.
- *Linear*: a linear layer to map the flattened activations to the logits.

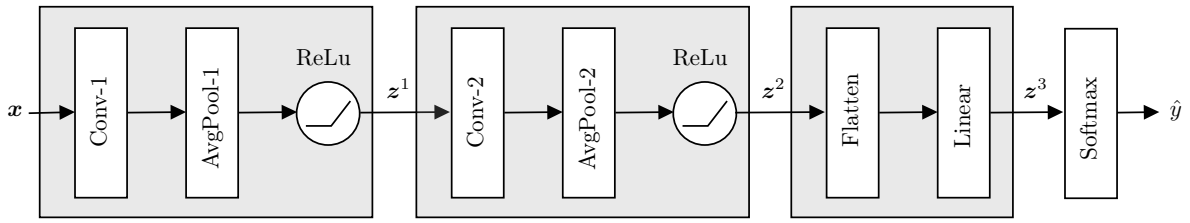

Figure 2: Convolutional neural network architecture.

1. (5 points) What is the *key advantage* of using pooling layers in a neural network? What differences would you expect between using a max pooling layer in comparison to an average pooling layer?

   Solution: Pooling layers provide invariance to the neural network, making the model more robust to small variations in the input, and increasing the receptive field of neurons in the deeper layers of the network.

   Max pooling layers select the maximum value for each patch of the feature map, while average pooling layers calculate the average of all values within the patch. While max pooling layers select the most prominent feature for each patch of the feature map, average pooling layers smoothen out all the feature values within the patch.

2. (5 points) How many trainable weights do each of the *Conv-1*, *AvgPool-1*, *Conv-2* and *AvgPool-2* layers of the neural network have? Do not forget to take into account bias terms.

   Solution: Layer *Conv-1*: num. weights = num. of filters × ((kernel width × kernel height × num. channels) + bias) = $8 \times ((5 \times 5 \times 1) + 1)$.

   Layer *Conv-2*: num. weights = num. of filters × ((kernel width × kernel height × num. channels) + bias) = $16 \times ((4 \times 4 \times 8) + 1)$.

   Layers *AvgPool-1* and *AvgPool-2* do not have trainable weights.

3. (10 points) A student implemented the neural network depicted in Fig. 2 in Pytorch, as seen below. However, when running the code the student gets an error due to mismatching dimensions when performing the dot product between two matrices. Given the error obtained by the student, fix the code provided below.

```
nn.Sequential(
    nn.Conv2d(in_channels=1, out_channels=8, kernel_size=5,
              stride=1, padding=0),
    nn.AvgPool2d(kernel_size=4, stride=4, padding=0)
    nn.ReLU(),
    nn.Conv2d(in_channels=8, out_channels=16, kernel_size=4,
              stride=1, padding=1),
    nn.AvgPool2d(kernel_size=2, stride=2, padding=0)
    nn.ReLU(),
    nn.Flatten(),
    nn.Linear(120, 10)
)
```

Solution: The input dimensions are $32 \times 32 \times 1$ (grayscale image).

The output of layer *Conv-1*, is of size $28 \times 28 \times 8$, where 8 is the number of channels of the convolutional layer and $\frac{\text{input dim.}-\text{kernel size}}{\text{stride}} + 1 = \frac{32-5}{1} + 1 = 28$.

The output of layer *AvgPool-1*, is of size $7 \times 7 \times 8$, where 8 is the number of channels (the same number of input channels of the layer) and $\frac{\text{input dim.}-\text{kernel size}}{\text{stride}} + 1 = \frac{28-4}{4} + 1 = 7$.

The output of layer *Conv-2*, is of size $6 \times 6 \times 16$, where 16 is due to the number of kernels. First, we pad the input of the layer yielding size $9 \times 9 \times 8$. Second, this input is convoluted with the kernels yielding an output dimension of $\frac{\text{input dim.}-\text{kernel size}}{\text{stride}} + 1 = \frac{9-4}{1} + 1 = 6$.

The output of layer *AvgPool-2*, is of size $3 \times 3 \times 16$, where 16 is the number of channels (the same number of input channels of the layer) and $\frac{\text{input dim.}-\text{kernel size}}{\text{stride}} + 1 = \frac{6-2}{2} + 1 = 3$.

Thus, after the flattening operation we obtain a vector of dimension $3 \times 3 \times 16 = 144$. Hence, the linear layer of the network should be initialized as `nn.Linear(144, 10)`.

4. (5 points) It should be possible to replace the last block of our CNN (i.e. the Flatten and Linear layers) with convolutions and still project to $\mathbb{R}^{10}$ to obtain z^3, and thus still be able to apply the *Softmax* layer and obtain class probabilities. Describe the convolutional layer that should be applied to be able to project to the output dimensions without needing a linear layer.

Solution:

Solution v1: The output of the second pooling layer *AvgPool-2* is $3 \times 3 \times 16$, hence we know that the input dimension of the convolution layer would be 16. Thus, if we apply a kernel with size 3×3 and with 10 output channels (filters) we will obtain outputs of dimension $1 \times 1 \times 10$.

Solution v2: The output of the second pooling layer *AvgPool-2* is $3 \times 3 \times 16$, hence we know that the input dimension of the convolution layer would be 16. If we apply a kernel with size 1×1 and with 10 output channels (filters) we will obtain outputs of dimension

$3 \times 3 \times 10$. If we then apply a third pooling layer *AvgPool-3* of size 3×3 we will obtain outputs of dimension $1 \times 1 \times 10$.

You can use this space as draft.

Problem 2: Transformer Model for Deduplicating and Sorting (25 points)

Geoff works as a data engineer at a startup. He needs to deduplicate and sort some data. His boss told him to use transformers because he heard they can solve any problem. So Geoff decides to implement a small decoder-only transformer-based model to solve this task.

His model uses only a single self-attention layer with a single attention head, without any feedforward layers or residual connections. The self-attention layer has the following parameters:

$$\boldsymbol{W}_Q = \boldsymbol{W}_K = \boldsymbol{W}_V = \begin{bmatrix} 0 & 3 \\ 1 & 2 \\ 2 & 1 \\ 3 & 0 \end{bmatrix}, \quad \boldsymbol{W}_O = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}.$$

The vocabulary is $\{\texttt{A}, \texttt{B}, \texttt{C}, \texttt{D}, \langle\text{START}\rangle, \langle\text{STOP}\rangle\}$ and the model uses the following embedding matrix $\boldsymbol{E} \in \mathbb{R}^{6\times 4}$:

$$\boldsymbol{E} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \# \texttt{A} \\ \# \texttt{B} \\ \# \texttt{C} \\ \# \texttt{D}. \\ \# \langle\text{START}\rangle \\ \# \langle\text{STOP}\rangle \end{matrix}.$$

Geoff wants to train the model to, given a sequence of symbols (where some symbols can be repeated), output a sequence with all duplicates removed and where the symbols appear sorted in lexicographic order. For example, the input sequence `D A A` should be mapped to the output `A D`.

1. (5 points) Geoff decided not to use any positional encodings. Was this a wise decision? Justify.

   Solution: It was a wise decision, since for this particular problem the output is invariant to the order by which the inputs are presented. For example, "`D A A`", "`A D A`", "`A A D`" should all map to the same output `A D`. Without positional encodings the transformer treats the input as a "bag of symbols" ignoring the sequential structure, which is a good inductive bias for this problem.

2. (8 points) Geoff wants to learn the model by minimizing the cross-entropy loss using teacher forcing. He decides to try something new in the attention mechanism: Let $\text{softmax}(\boldsymbol{s}/T)$ denote the softmax transformation of a vector $\boldsymbol{s} \in \mathbb{R}^d$ with temperature T. Normally, transformers use scaled dot product attention with $T = \sqrt{d}$, but in this model Geoff decides to use "zero temperature" ($T \to 0^+$) which corresponds to argmax. For example, $\lim_{T\to 0_+} \text{softmax}([1, -2, 5, 0]^\top/T) = [0, 0, 1, 0]^\top$ and $\lim_{T\to 0_+} \text{softmax}([5, -2, 5, 0]^\top/T) = [\frac{1}{2}, 0, \frac{1}{2}, 0]^\top$.

   Compute the attention matrix for the training example "`D A A` $\langle\text{START}\rangle$ `A D` $\langle\text{STOP}\rangle$". Include all the steps in your calculation.

   Note: Remember that this is a decoder-only model, so you need to use **causal masking**.

Solution: The input embeddings are

$$\boldsymbol{X} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{matrix} \# \texttt{D} \\ \# \texttt{A} \\ \# \texttt{A} \\ \# \langle \text{START} \rangle \\ \# \texttt{A} \\ \# \texttt{D} \\ \# \langle \text{STOP} \rangle \end{matrix} .$$

We obtain

$$\boldsymbol{Q} = \boldsymbol{K} = \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_Q = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 3 \\ 1 & 2 \\ 2 & 1 \\ 3 & 0 \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 0 & 3 \\ 0 & 3 \\ 0 & 0 \\ 0 & 3 \\ 3 & 0 \\ 0 & 0 \end{bmatrix}$$

We have

$$\boldsymbol{Q}\boldsymbol{K}^\top = \begin{bmatrix} 3 & 0 \\ 0 & 3 \\ 0 & 3 \\ 0 & 0 \\ 0 & 3 \\ 3 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 3 & 3 & 0 & 3 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 9 & 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & 9 & 9 & 0 & 9 & 0 & 0 \\ 0 & 9 & 9 & 0 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 9 & 9 & 0 & 9 & 0 & 0 \\ 9 & 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

With causal argmax attention, we obtain

$$\boldsymbol{P} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 & 0 & 0 & 0 \\ 1/4 & 1/4 & 1/4 & 1/4 & 0 & 0 & 0 \\ 0 & 1/3 & 1/3 & 0 & 1/3 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 1/2 & 0 \\ 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 \end{bmatrix} .$$

3. (5 points) Compute the cross-entropy loss value for the same training example (note that the loss is applied only to the output tokens "`A D` ⟨STOP⟩"). Assume that the representations after the self-attention layer are multiplied by $\boldsymbol{W}_O$ and then converted to logits by multiplying by $\boldsymbol{E}^\top$ (the transpose of the embedding matrix written above).

Note: Assume that the attention matrix obtained in the previous exercise is

$$\boldsymbol{P} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 & 0 & 0 & 0 \\ 1/4 & 1/4 & 1/4 & 1/4 & 0 & 0 & 0 \\ 0 & 1/3 & 1/3 & 0 & 1/3 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 1/2 & 0 \\ 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 \end{bmatrix}.$$

Solution: The output of the self-attention layer is

$$\boldsymbol{Z} = \boldsymbol{P}\boldsymbol{V} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 & 0 & 0 & 0 \\ 1/4 & 1/4 & 1/4 & 1/4 & 0 & 0 & 0 \\ 0 & 1/3 & 1/3 & 0 & 1/3 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 0 & 1/2 & 0 \\ 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & 3 \\ 0 & 3 \\ 0 & 0 \\ 0 & 3 \\ 3 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 0 & 3 \\ 0 & 3 \\ 3/4 & 3/2 \\ 0 & 3 \\ 3 & 0 \\ 6/7 & 9/7 \end{bmatrix}.$$

Multiplying by $\boldsymbol{W}_O$ leads to

$$\boldsymbol{R} = \boldsymbol{Z}\boldsymbol{W}_O = \begin{bmatrix} 3 & 0 \\ 0 & 3 \\ 0 & 3 \\ 3/4 & 3/2 \\ 0 & 3 \\ 3 & 0 \\ 6/7 & 9/7 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 3 & 0 & 3 \\ 3 & 0 & 3 & 0 \\ 3 & 0 & 3 & 0 \\ 3/2 & 3/4 & 3/2 & 3/4 \\ 3 & 0 & 3 & 0 \\ 0 & 3 & 0 & 3 \\ 9/7 & 6/7 & 9/7 & 6/7 \end{bmatrix}.$$

To compute the loss, only the bottom three representations matter. Multiplying by $\boldsymbol{E}^\top$ we get the logits

$$\begin{aligned} \boldsymbol{z}_5^\top &= [3, 0, 3, 0]\boldsymbol{E}^\top = [3, 0, 3, 0, 0, 0] \\ \boldsymbol{z}_6^\top &= [0, 3, 0, 3]\boldsymbol{E}^\top = [0, 3, 0, 3, 0, 0] \\ \boldsymbol{z}_7^\top &= [9/7, 6/7, 9/7, 6/7]\boldsymbol{E}^\top = [9/7, 6/7, 9/7, 6/7, 0, 0] \end{aligned}$$

Therefore,

$$\begin{aligned} P(y_5 = \texttt{A}) &= \exp(3)/(2\exp(3) + 4) = 0.4547 \\ P(y_6 = \texttt{D}) &= \exp(3)/(2\exp(3) + 4) = 0.4547 \\ P(y_7 = \langle\text{STOP}\rangle) &= 1/(2\exp(9/7) + 2\exp(6/7) + 2) = 0.0717. \end{aligned}$$

The loss is

$$L = -\log P(y_5 = \texttt{A}) - \log P(y_6 = \texttt{D}) - \log P(y_7 = \langle\text{STOP}\rangle) = 4.2114.$$

4. (4 points) John decides to add a second attention head with projection matrices $\boldsymbol{W}'_Q = \boldsymbol{W}_Q$, $\boldsymbol{W}'_K = \boldsymbol{W}_K$, $\boldsymbol{W}'_V = \boldsymbol{W}_V$ (i.e. the second attention head is identical to the first one). He also changes $\boldsymbol{W}_O$ to be the 4-by-4 identity matrix. Repeat the previous exercise.

Solution: We would concatenate two copies of the matrix $\boldsymbol{Z}$ as above and would obtain the same $\boldsymbol{R}$, so the result would be identical.

5. (3 points) After running a few epochs of gradient descent, John notices that the model is not learning anything. The projection matrices $\boldsymbol{W}_Q$, $\boldsymbol{W}_K$, $\boldsymbol{W}_V$ never change. What could be the problem? Will the problem be fixed if he switches to the usual softmax attention with temperature $T = \sqrt{d}$?

   Solution: The argmax attention has zero gradient, therefore the projection matrices are never updated in the gradient backpropagation algorithm. The problem would be fixed by using the usual softmax attention with temperature $T = \sqrt{d}$.